COMP4102 COURSE PROJECT: HANDY KEYS🤟

Date: 2022-04-12

Author: Erik Iuhas 101076512

ABSTRACT

This project utilizes computer vision and machine learning to classify different hand gestures and create key inputs depending on the classification made. In this project, the machine learning model employed is a modified VGG network running within a Kivy app. The project's findings indicated that a classification network could identify and differentiate finger position differences. After identifying the hand gesture, the network displays the hand gesture of the user, and in future implementations of the project, it could be used to activate a keystroke.

INTRODUCTION

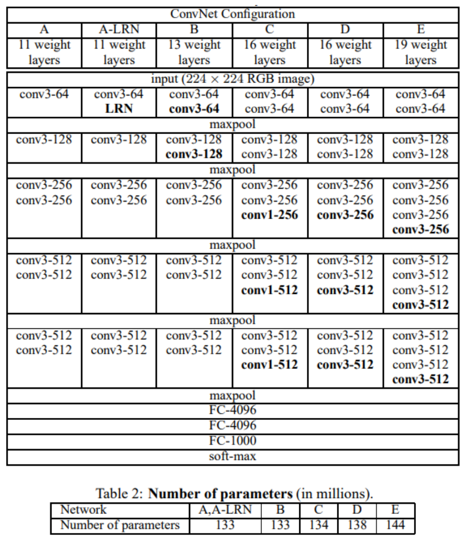
As an artist, I often get wrist pain in my free time due to my wrist sitting on my keyboard for hours on end inputting key inputs. Due to that, my passion for machine learning and computer vision compelled me to create a neural network that could detect my hand gestures for keyboard inputs instead. Because my project was self-motivated, there were challenges in me attempting to find a dataset that matched the qualities I needed for hand gestures. Because of that, I was forced to create my dataset. I know that the network will likely overfit my hands and setting, failing to generalize for other people's hands. Another difficulty I faced was creating a smaller network that could correctly classify my hands and learn from the smaller dataset without overfitting.

BACKGROUND

There already exist other projects which look to utilize machine learning to track hand gestures and other movements; an example of such a project is the MediaPipe Hands library[1]. In this library, they can follow 3D hand landmarks, such as the critical points on the hand being the tips of the fingers and knuckles and then identify the angle and position of all other fingers.

At the beginning of the project, I considered such a solution to recognize hand gestures, but the prospect proved to be too tricky as media pipe does not provide the methodology and the network, they used to train the hand gesture detector. It would also require me to create my database of complex annotations, which is outside of my time budget. I considered other machine learning projects that could support my endeavours.

The research paper titled "Very deep convolutional networks for large-scale image recognition"[2] provided me with grounds that could be used to structure and train my own convolutional neural network. The leading network named in the research paper was a VGG network with varying levels of depth ness. The VGG is a type of convolutional network and does not require as many layers as more advanced networks such as ResNet50's that require residual blocks to preserve image features.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Figure 2. ConvNet Configurations.

For this project, I looked forward to creating a network that could utilize a ConvNet/VGG network as a base while attempting to increase the performance on the limited dataset I made myself.

METHODOLOGY

Dataset Generation

Because my project was self-driven, I had to create my dataset consisting of my hand gestures to train the neural network. To generate and annotate my dataset, I set out to record a video of myself using my web camera. Before recording, I set out to give myself rules in making the dataset not to overcomplicate the project; limit the number of hand gestures to seven and one class for nothing; doing so will allow for the network to learn easier and prevent me from having to create too much of my dataset. The second was to create a diverse dataset with my own hands, with different lighting and backgrounds to teach the network the essential gestures.

Knowing this, I set out to record short videos where I position my hand at various angles and distances from the camera; afterwards, I process the videos into a square frame and export the frames of each image into a training set and validation set using a python script that utilized OpenCV. The split I settled on for the dataset was an 80/20 split; I had 99 photos in my training set and 21 photos in the validation set, for each gesture. Including all the images together it amounts to 960 photos; 792 in the training set and 168 in the validation set. While the size of the training set seems small, I will detail how I worked around it to increase my dataset size synthetically. An example of the eight classifications is available in the table below.

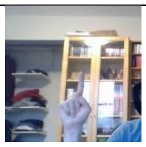
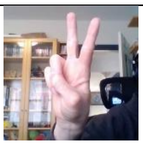
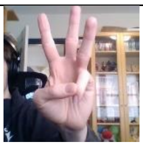
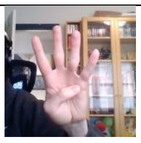
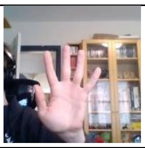
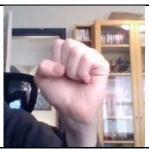
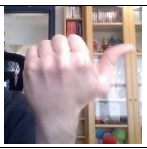
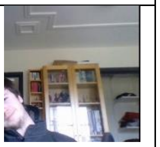
	Hand 1	Hand 2	Hand 3	Hand 4	Hand 5	Hand 6	Hand 7	None
Ex.								

Table 1. Contains photos from the validation set of hand gesture examples

Training a CNN

The computer vision portion is due to my dataset's convolutional neural network used to train. As mentioned earlier, I based the initial network design on a VGG network provided in the research paper. Specifically, configuration A in Figure 2. This was the smallest of the provided networks and would be an excellent starting ground for the project.

Before training on the network, it was essential for me to initialize the database for usage for training. To do this, I used Google Drive and mounted it within my Google Colab. Doing so allowed me to train models and test different training variables quickly. Since the training dataset used for the models was small, I needed to apply a machine learning technique to create synthetic data. To do this, I applied horizontal flips to all images and allowed for images to be shifted horizontally and vertically by a scale of 0.05 and 0.1. Doing so effectively made the dataset larger and should help with regularizing the training process for the CNNs. After initializing the datasets, I could construct the different architectures tested and used in the project.

Architectures

VGG

For the base VGG, I followed the report's structure and recreated created a sequential network using TensorFlow and Keras. The network architecture is large and has about 143 million trainable parameters. I made a sequential model based on the structure seen in **Figure 2**; configuration A. **Figure 3** illustrates the network's architecture below.

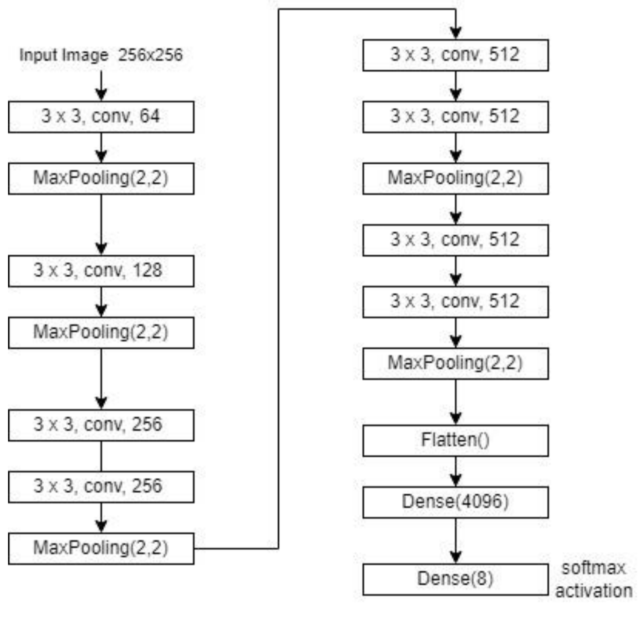


Figure 3. Base VGG Network Architecture

VGG MODIFIED – TINYHAND

My modified VGG network, which I named **TinyHand**, was made to increase the regularization while reducing the number of input variables, preventing the network from overfitting. Doing so will allow it to generalize better on new data such as the validation set or when used outside of testing. To do this, I modified the depth of each convolution batch within the network and introduced dropout layers. Reducing the depth limited the number of variables for the network; this will help when deploying the HandyKey App as it doesn't require as much computing power from the local machine as it would make calls to the model at least four times a second. The dropout layers operate by selecting specific variables from the previous network and turning them off so that other layers learn more object features, preventing one layer from overfitting and making the network over-reliant on that layer. By doing so, I reduced the trainable parameters down to 35 million. I've also included additional convolution layers. With the network's reduced size, it should make it easier for local devices to run it frequently. You can see the final architecture of my network in Figure 4.

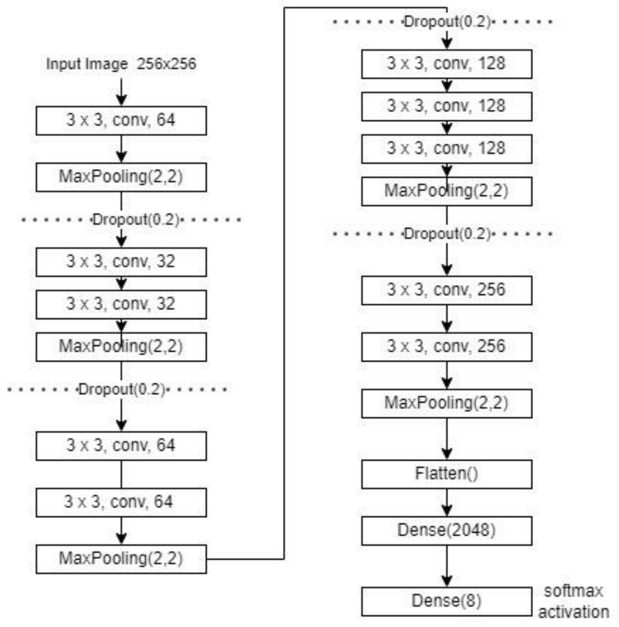


Figure 4. TinyHand Network Architecture

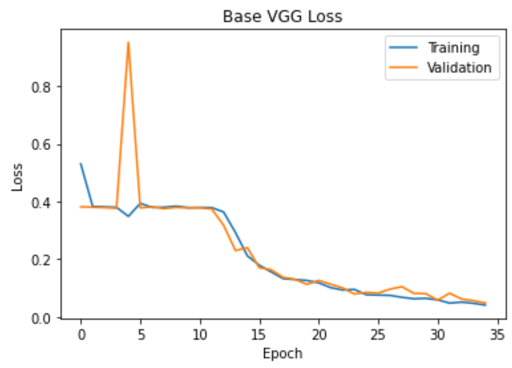
App Development

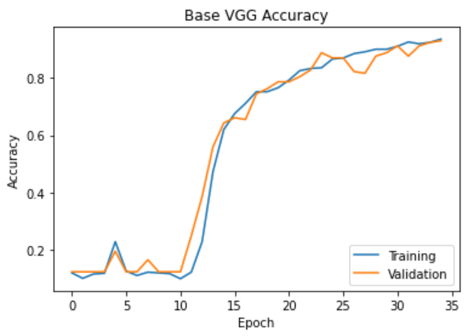
After developing the architecture, I created an app in python using the Kivy library. Doing so allowed me to load the trained model TinyHand and test it as a real-life application. Doing so showed me that both models failed to perform well with new data, and I quickly realized that I had overfitted my initial dataset and that more data had to be collected. The current training dataset size of 792 was initially 600 and only consisted of one lighting scene with one background. This led me to generate more images for the dataset. The following training results are made following the increase in dataset size.

RESULTS

VGG

Training the base VGG model from the research paper achieved a desirable result around eight epochs in training. The model slowly started to lower the loss as activity continued, but it was unlikely that the network could improve performance and likely begin overfitting to the dataset. It could be seen towards the end, around 30-35 epochs, that it was starting to go down only slightly and would likely no longer improve.





Overall, the network had a performance of 93.5% Accuracy and 91.54 Recall on the Training set and 92.86% Accuracy and 91.67% Recall on the Validation set.

The performance of this network gave me a good starting ground to plan and create my next model, TinyHand.

TinyHand

The results for training tiny hands at first were non-ideal because the network failed to properly lower its loss, and in the case it did, it was unable to perform well on new data. I believe this to be because when I was changing the values of the layers and making the network smaller, I reduced the first layer to 32, which greatly affected the network's performance. The results of that training can be seen below in Figure 6. I had to train about six different versions of TinyHand before settling on the current design mentioned in the architecture section.

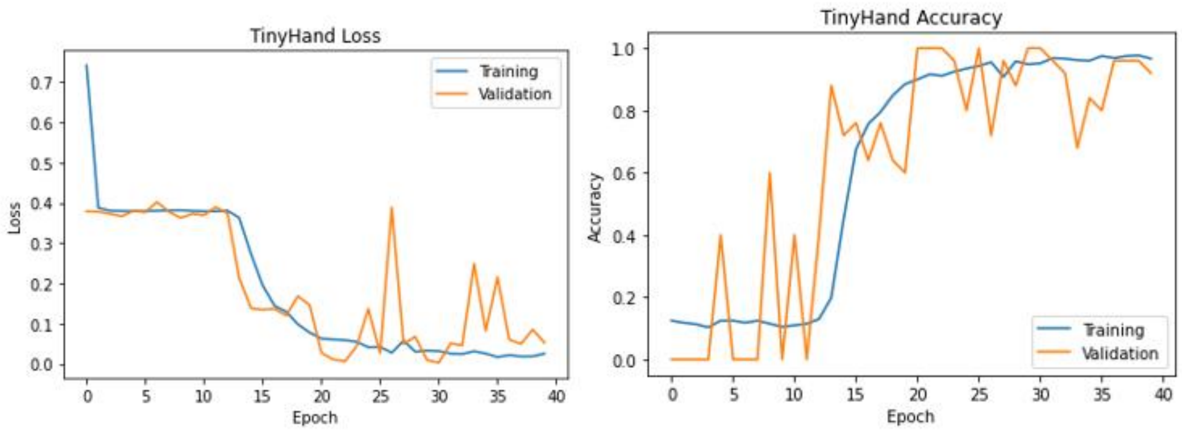


Figure 6. Initial training of TinyHand Model

The training results for the final TinyHand Model can be seen below in Figure 7.

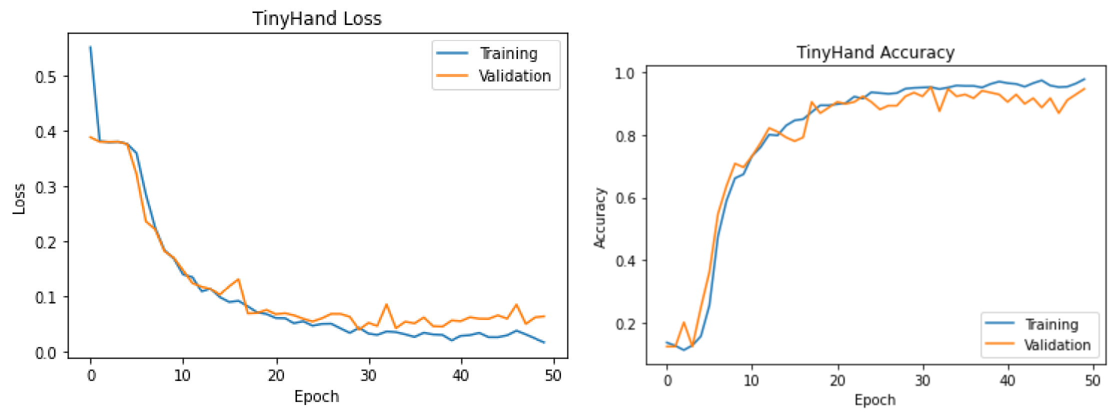


Figure 7. Final Training results of TinyHand Model

The results of the TinyHand Model trained on the complete dataset were promising as it was able to stare gradient decent sooner than the previous VGG model, implying the network could quickly learn the features of the hand. The final accuracy and recall for the network were **97.73% and 97.35%** for training and **94.64% and 94.05%** for validation. As seen in the accuracy, the network at the end increased accuracy for the training and validation. This implies that the network isn't overfitting on the training data and should be able to generalize due to the dropout layers I added to the configuration.

TinyHand App

I found when developing the App in Kivy that the network had a hard time detecting hands when the background was more complex and consisted of different lighting than was present in the training set; this is to be expected with the dataset size I had for the project.

The network was able to perform best when it came to a dark lighting scene because it helped clearly show the hands while ignoring the dark background. The results of the TinyHand App can be seen below in Figure 8, with three hand gesture examples.

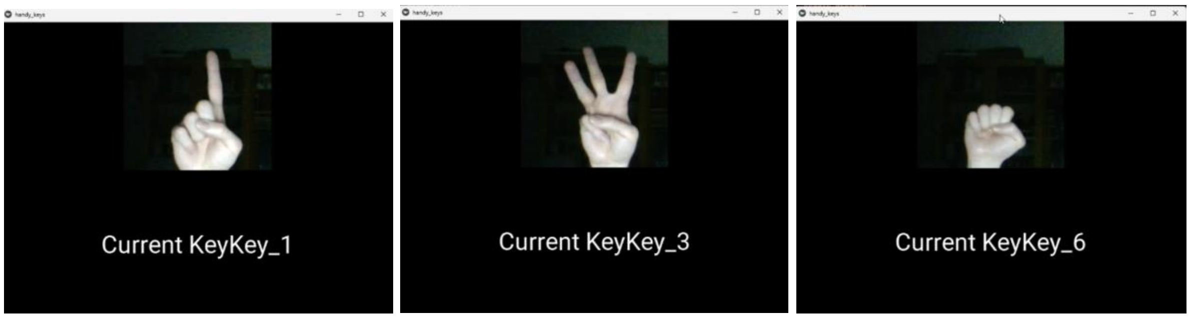


Figure 8. Three examples in the HandyKeys App being detected in a dark setting

In the App, I was taking samples every four seconds and often, the model was not performing perfectly, so I did not implement the keystroke feature as it would likely lead to the wrong keystrokes being inputted. By displaying the "Current Key" in the app as seen in the demo provided it could be implied that a keystroke would be pressed every time the label at the bottom of the window updates.

DISCUSSION

Overall, after training my own network, I believe I found the limits of using a classification network trained on a dataset alone, even when I create my dataset with my hands. The TinyHand model had had trouble with gestures with different backgrounds. Due to this, I believe that increasing my dataset to be more diverse could improve the performance of the model, as I first mentioned in the Results section.

The App I developed within Kivy may also benefit from a better gesture recognition library rather than a classification solution, as there is no 100% guarantee that the gesture is correct. If the model quickly mistakes the hand gesture, if a user used the App, it would cause the user to input the wrong key, counteracting the application's entire use-case. I also found that the network performed best in a dark setting; I believe this to be because the background is less detailed and the hand becomes more apparent, making it easier for the TinyHand model to detect the hand gestures.

Overall, I believe that with enough time, HandyKeys could provide a convenient solution to artists but not with the current TinyHand model and App developed in this project.

REFERENCES

[1] "Hands," *mediapipe*. [Online]. Available: https://google.github.io/mediapipe/solutions/hands.html. [Accessed: 12-Apr-2022].

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv.org*, 10-Apr-2015. [Online]. Available: https://arxiv.org/pdf/1409.1556.pdf [Accessed: 11-Apr-2022].